

Zero-Overhead Loop Control for Tightly Coupled Processor Arrays (TCPAs)

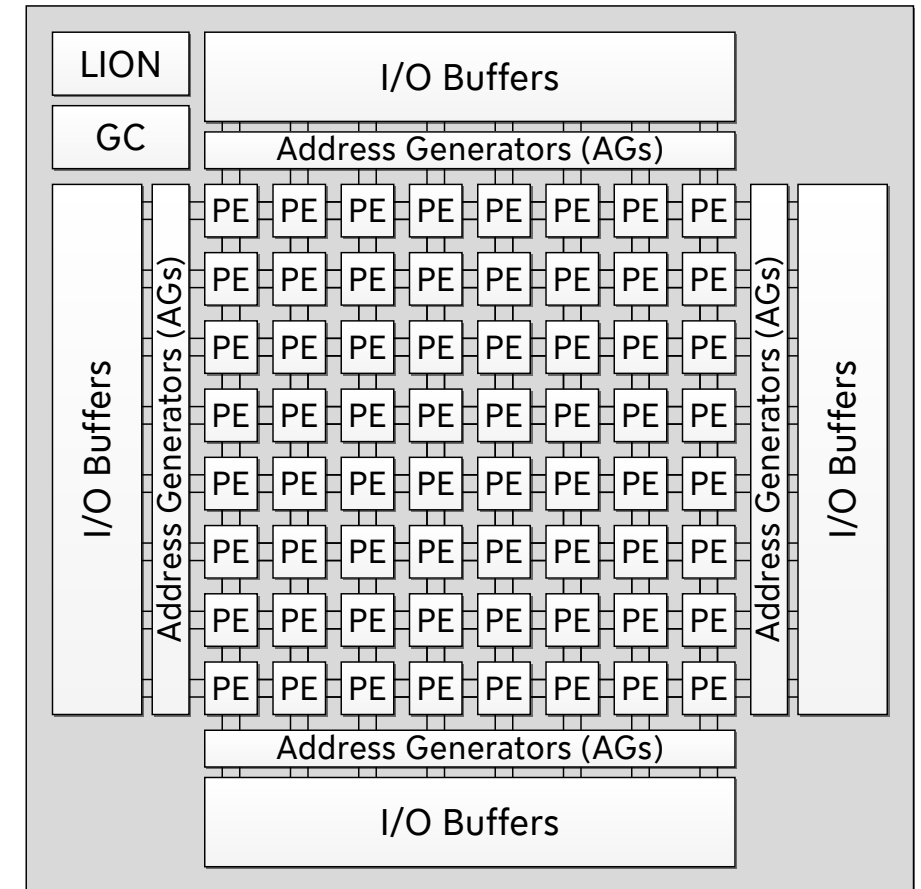
Reducing control complexity through global, polyhedral loop control

Dominik Walter, Frank Hannig, Jürgen Teich

Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

37th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)
London, September 2-4, 2026

- Many workloads are dominated by **nested loops**:
 - Linear algebra (GEMM, TRSM, ...)
 - Signal & image processing (convolutions, filters)
 - Scientific computing (stencils, solvers)
- Nested loops expose **substantial parallelism**:
 - Data-level parallelism and software pipelining across iterations
 - Instruction-level parallelism within iterations
- Processor arrays exploit this parallelism with:
 - 2D grid of interconnected processing elements (PEs)
 - Each PE is programmable/reconfigurable
 - PEs communicate via *reconfigurable direct PE-to-PE links*
- **Challenge**: Loop control can dominate execution time.



D. Kissler et al. "A Highly Parameterizable Parallel Processor Array Architecture". In: *FPT*. 2006, pp. 105–112.

F. Hannig et al. "Invasive Tightly-Coupled Processor Arrays: A Domain-Specific Architecture/Compiler Co-Design Approach". In: *TECS* 13.4s (2014), 133:1–133:29.

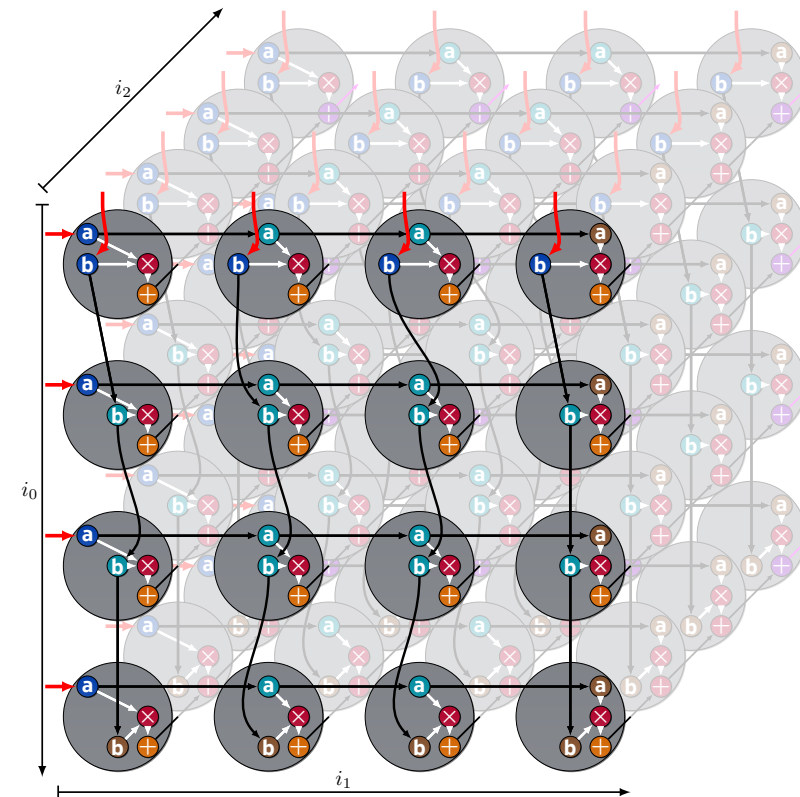
J. Teich et al. "Invasive Tightly-Coupled Processor Arrays". In: *Invasive Computing*. FAU University Press, 2022.

Loop Mapping

Example: matrix-matrix multiplication

Matrix-matrix multiplication

$$\begin{array}{llll} S_{a,1}: & a[i_0, i_1, i_2] & = & A[i_0, i_2] & \text{if } i_1 = 0 \\ S_{a,2}: & a[i_0, i_1, i_2] & = & a[i_0, i_1 - 1, i_2] & \text{if } i_1 > 0 \\ S_{b,1}: & b[i_0, i_1, i_2] & = & B[i_2, i_1] & \text{if } i_0 = 0 \\ S_{b,2}: & b[i_0, i_1, i_2] & = & b[i_0 - 1, i_1, i_2] & \text{if } i_0 > 0 \\ S_p: & p[i_0, i_1, i_2] & = & a[i_0, i_1, i_2] \cdot b[i_0, i_1, i_2] & \\ S_{s,1}: & s[i_0, i_1, i_2] & = & p[i_0, i_1, i_2] & \text{if } i_2 = 0 \\ S_{s,2}: & s[i_0, i_1, i_2] & = & p[i_0, i_1, i_2] + s[i_0, i_1, i_2 - 1] & \text{if } i_2 > 0 \wedge i_2 < N - 1 \\ S_C: & C[i_0, i_1] & = & p[i_0, i_1, i_2] + s[i_0, i_1, i_2 - 1] & \text{if } i_2 = N - 1 \end{array}$$

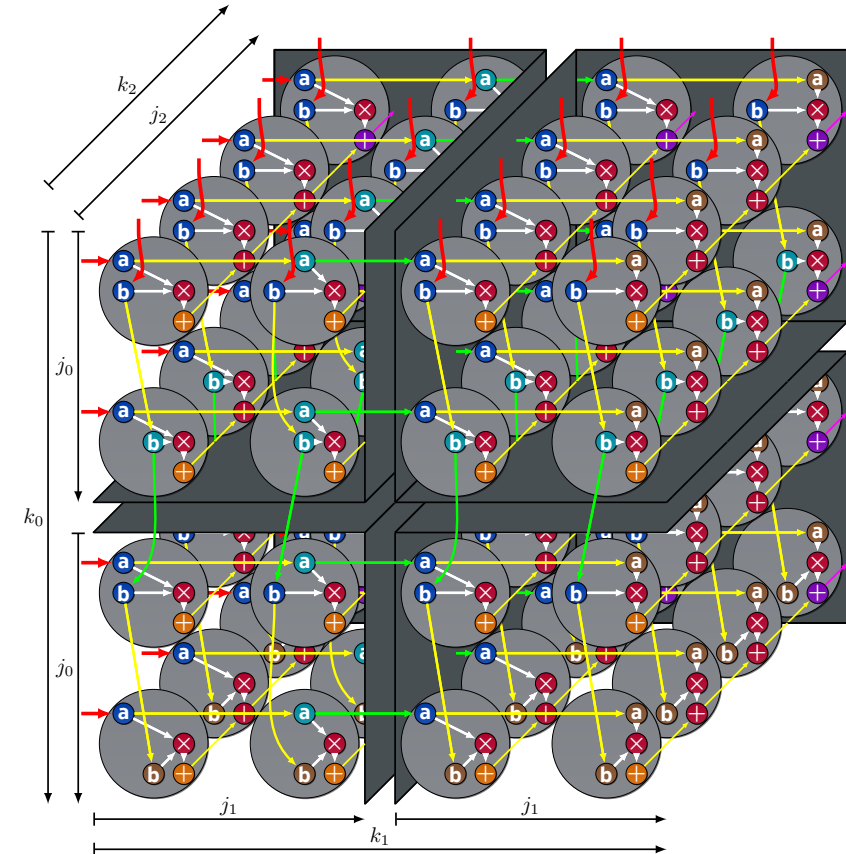


Loop Mapping

Example: matrix-matrix multiplication

Matrix-matrix multiplication

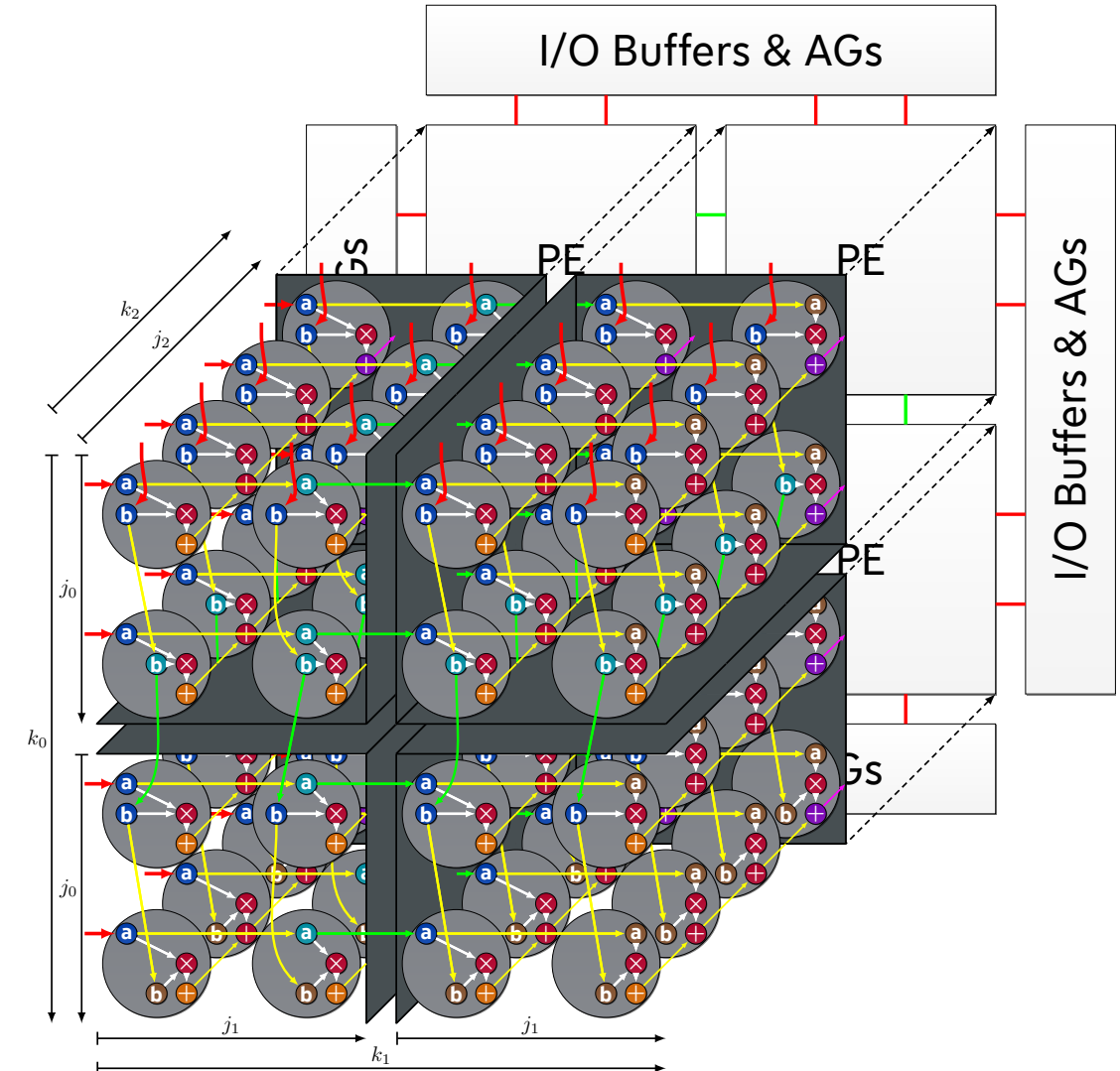
$$\begin{aligned}
 S_{a,1}: a[i_0, i_1, i_2] &= A[i_0, i_2] && \text{if } i_1 = 0 \\
 S_{a,2}: a[i_0, i_1, i_2] &= a[i_0, i_1 - 1, i_2] && \text{if } i_1 > 0 \\
 S_{b,1}: b[i_0, i_1, i_2] &= B[i_2, i_1] && \text{if } i_0 = 0 \\
 S_{b,2}: b[i_0, i_1, i_2] &= b[i_0 - 1, i_1, i_2] && \text{if } i_0 > 0 \\
 S_p: p[i_0, i_1, i_2] &= a[i_0, i_1, i_2] \cdot b[i_0, i_1, i_2] \\
 S_{s,1}: s[i_0, i_1, i_2] &= p[i_0, i_1, i_2] && \text{if } i_2 = 0 \\
 S_{s,2}: s[i_0, i_1, i_2] &= p[i_0, i_1, i_2] + s[i_0, i_1, i_2 - 1] && \text{if } i_2 > 0 \wedge i_2 < N - 1 \\
 S_C: C[i_0, i_1] &= p[i_0, i_1, i_2] + s[i_0, i_1, i_2 - 1] && \text{if } i_2 = N - 1
 \end{aligned}$$



Loop Mapping

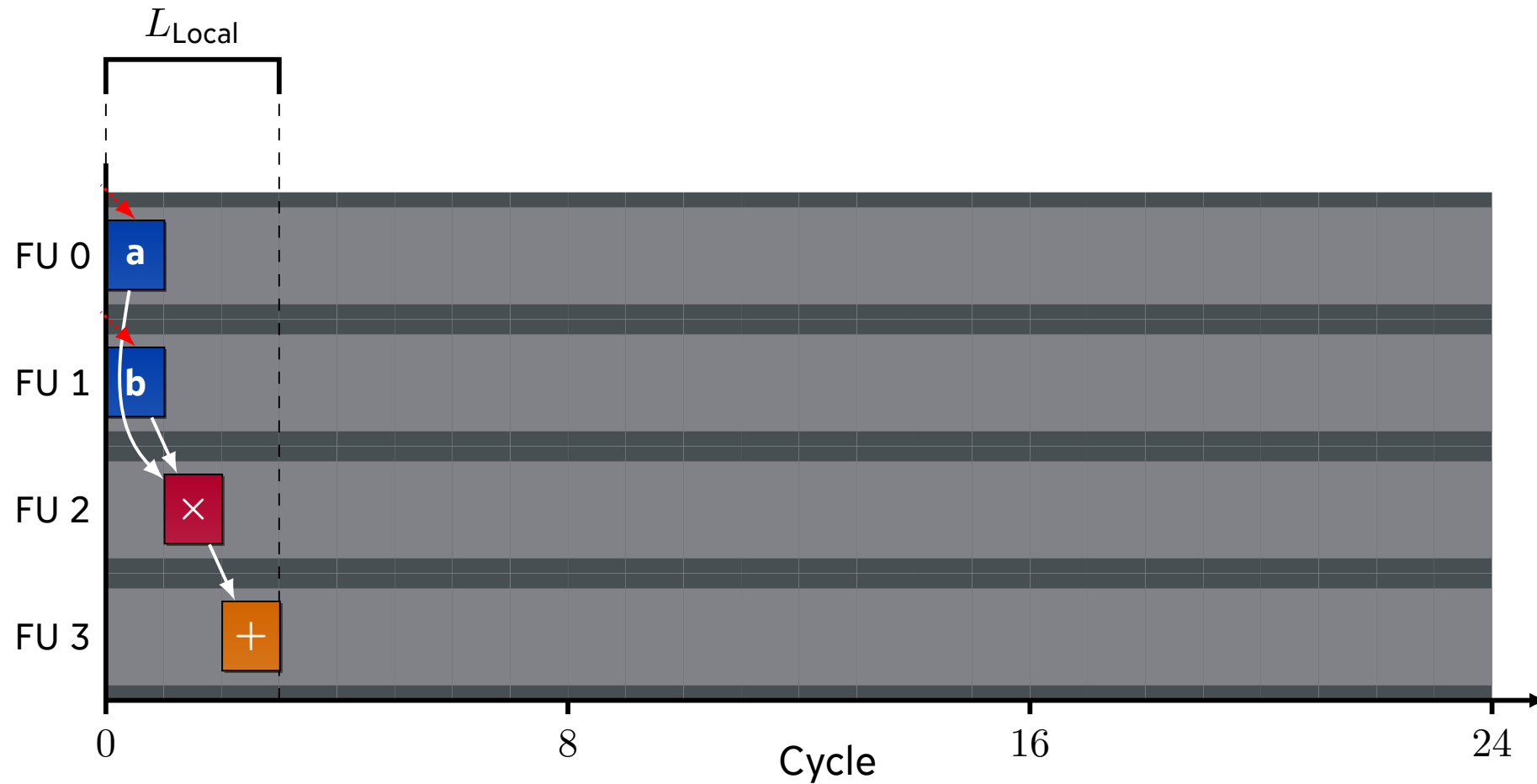
Example: matrix-matrix multiplication

Matrix-matrix multiplication

$$\begin{aligned}
 S_{a,1}: a[i_0, i_1, i_2] &= A[i_0, i_2] && \text{if } i_1 = 0 \\
 S_{a,2}: a[i_0, i_1, i_2] &= a[i_0, i_1 - 1, i_2] && \text{if } i_1 > 0 \\
 S_{b,1}: b[i_0, i_1, i_2] &= B[i_2, i_1] && \text{if } i_0 = 0 \\
 S_{b,2}: b[i_0, i_1, i_2] &= b[i_0 - 1, i_1, i_2] && \text{if } i_0 > 0 \\
 S_p: p[i_0, i_1, i_2] &= a[i_0, i_1, i_2] \cdot b[i_0, i_1, i_2] \\
 S_{s,1}: s[i_0, i_1, i_2] &= p[i_0, i_1, i_2] && \text{if } i_2 = 0 \\
 S_{s,2}: s[i_0, i_1, i_2] &= p[i_0, i_1, i_2] + s[i_0, i_1, i_2 - 1] && \text{if } i_2 > 0 \wedge i_2 < N - 1 \\
 S_C: C[i_0, i_1] &= p[i_0, i_1, i_2] + s[i_0, i_1, i_2 - 1] && \text{if } i_2 = N - 1
 \end{aligned}$$


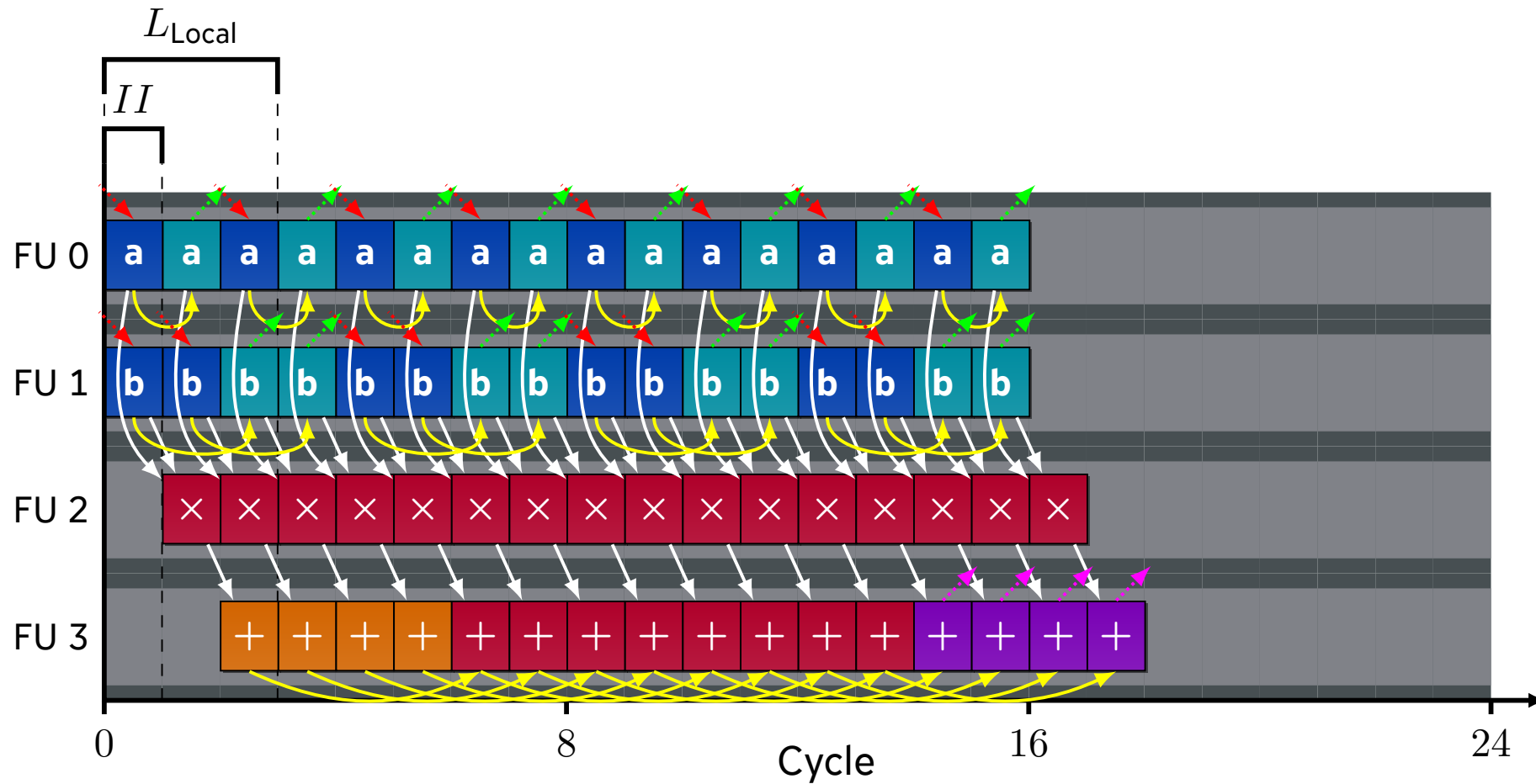
Local Scheduling

Scheduling one iteration across the array



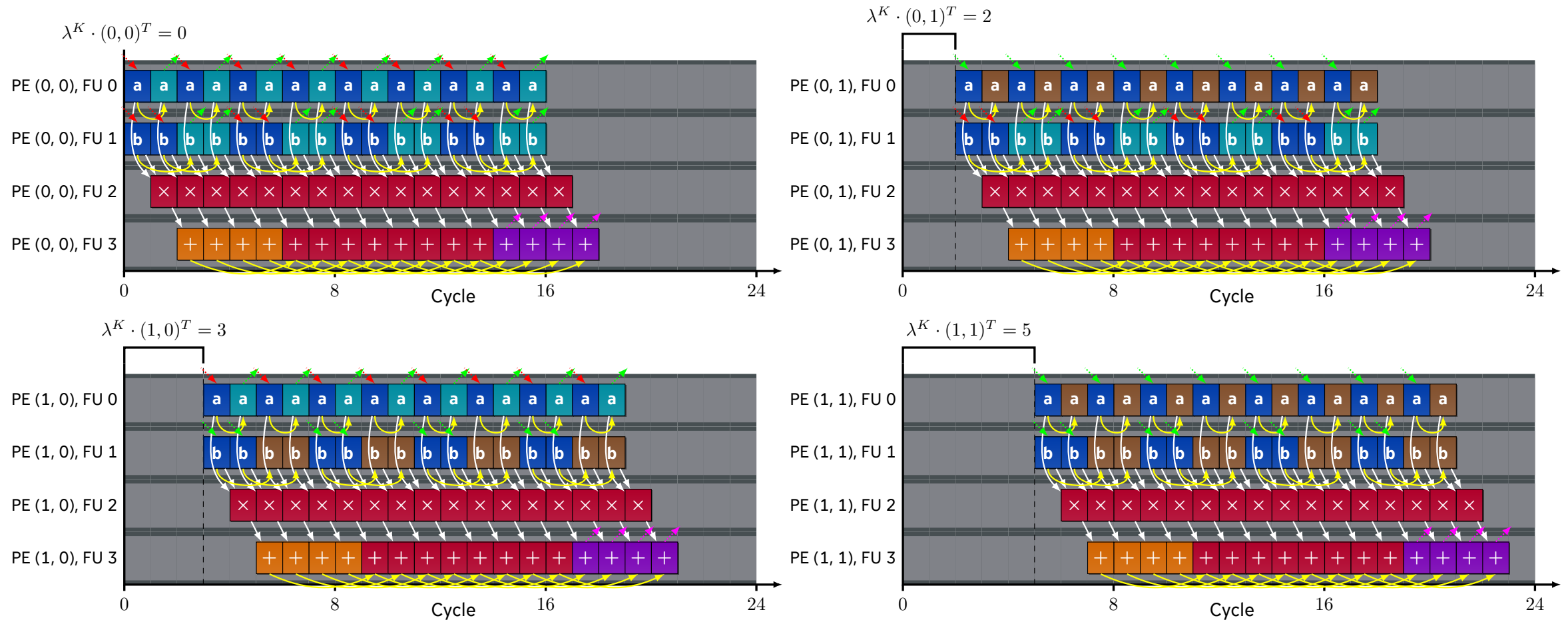
Local Scheduling

Scheduling one iteration across the array

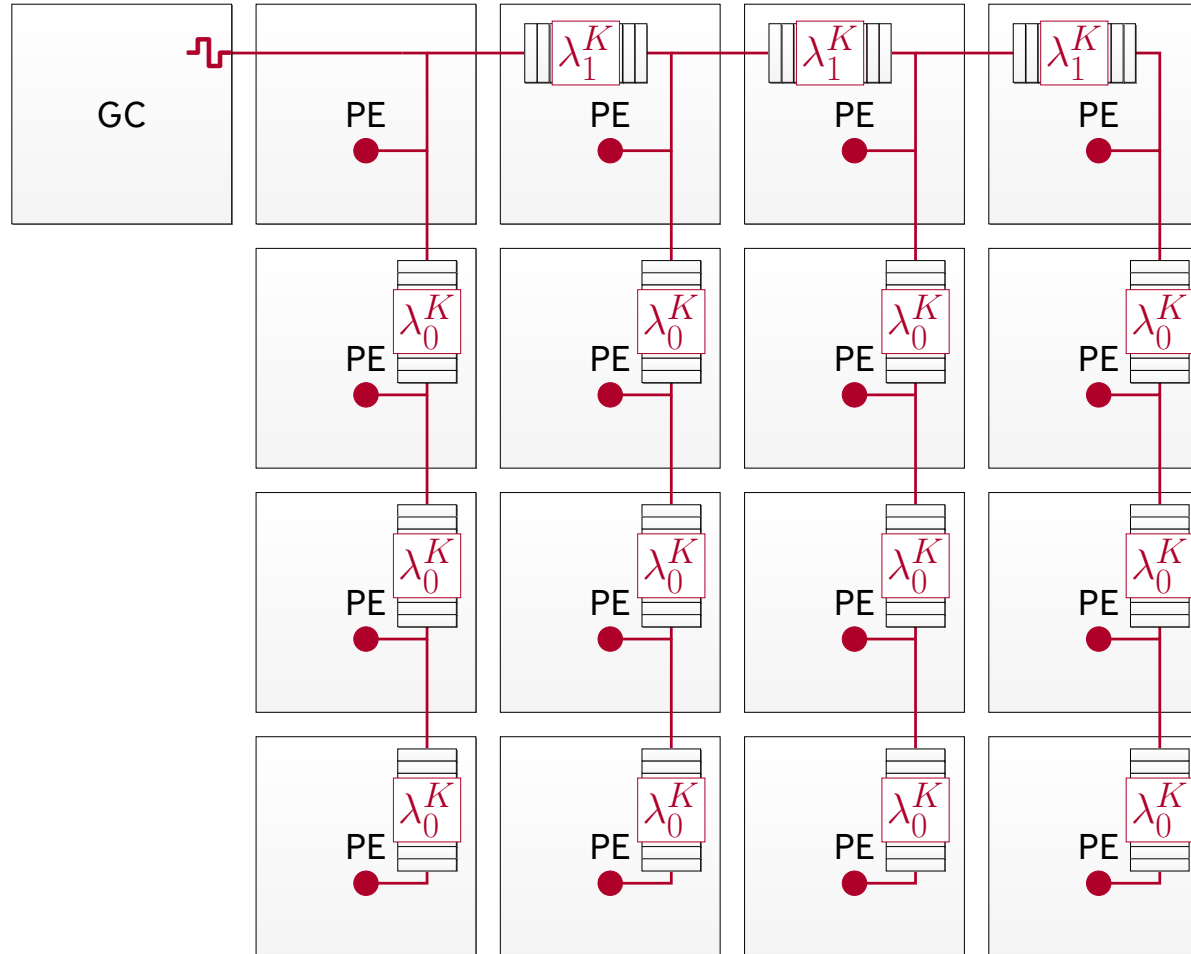


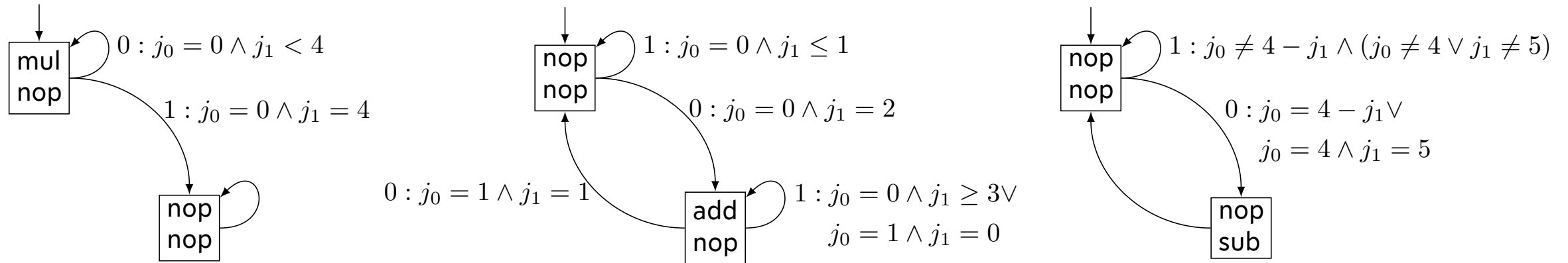
Global Scheduling

Repeating the schedule across tiled iterations



Overview



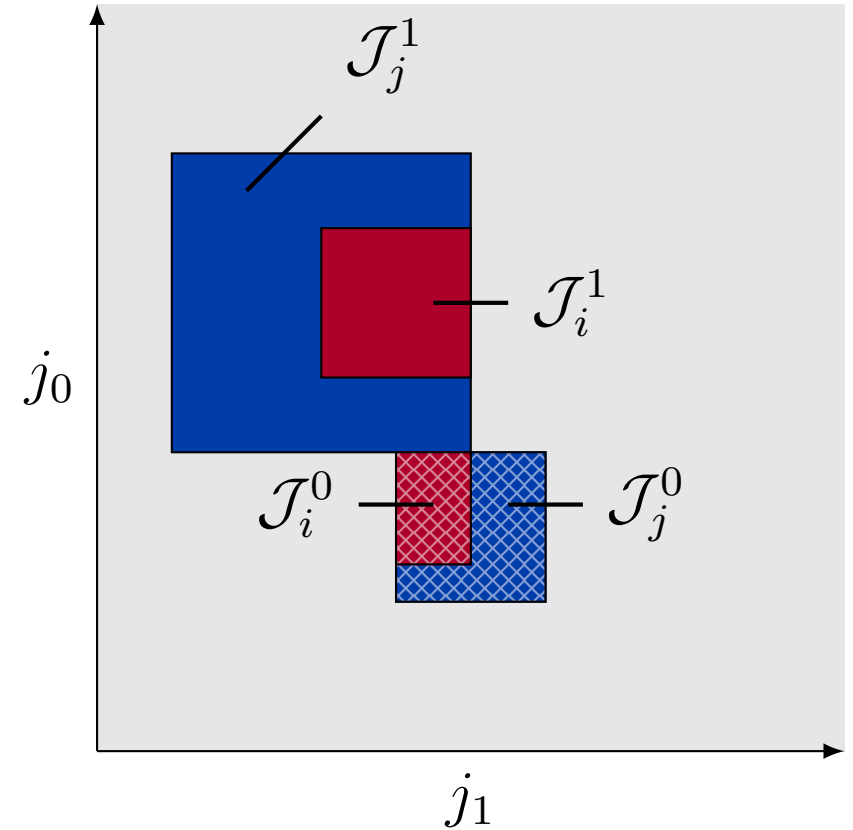


- **Nodes:** Program blocks, i.e., sequences of instructions
 - **Instruction:** Encodes an operation with its destination and source registers.
- **Edges:** Branches annotated with conditions over the intra-tile iteration space $\mathcal{I}$.
 - **Zero domain:** Iterations in $\mathcal{I}_i^0 \subseteq \mathcal{I}$ take the edge annotated with 0.
 - **One domain:** Iterations in $\mathcal{I}_i^1 \subseteq \mathcal{I}$ take the edge annotated with 1.

$$\Rightarrow \text{control condition } c_i = (\mathcal{I}_i^0, \mathcal{I}_i^1) \in \mathcal{C}$$

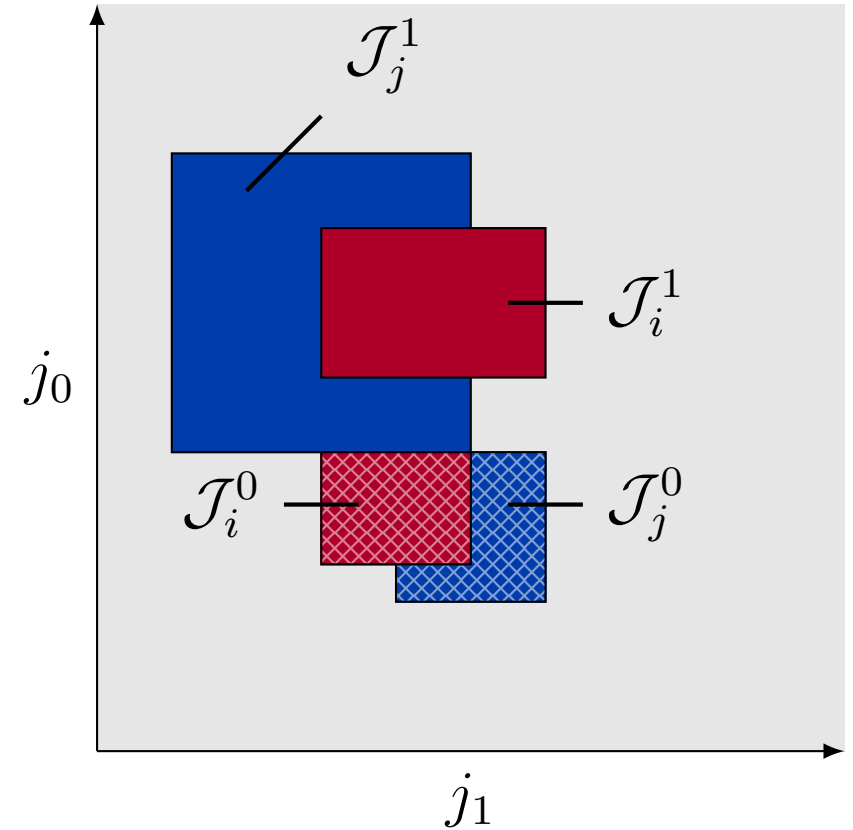
- Shown in red: control condition $c_i = (\mathcal{J}_i^0, \mathcal{J}_i^1)$
- Shown in blue: control condition $c_j = (\mathcal{J}_j^0, \mathcal{J}_j^1)$
- Evaluating c_j yields also c_i .

⇒ c_i is redundant and can be discarded.
⇒ An exhaustive search identifies all prime conditions.
⇒ The remaining conditions form $\mathcal{C}_{\text{prime}}$.



- Shown in red: control condition $c_i = (\mathcal{J}_i^0, \mathcal{J}_i^1)$
- Shown in blue: control condition $c_j = (\mathcal{J}_j^0, \mathcal{J}_j^1)$
- c_i does not contradict c_j .

⇒ Replace c_i and c_j with one unified condition c_k .
⇒ A greedy random-order search explores candidate pairs.
⇒ The smallest solution after 100 trials forms $\mathcal{C}_{\text{unified}}$.



The one domain $\mathcal{J}_i^1$ of each unified condition c_i is a union of polyhedra, i.e., a disjunction of conjunctions of loop-index comparisons, e.g.:

$$j_0 = 0 \wedge j_1 \geq 0 \wedge j_1 \leq 5 \quad \vee \quad j_0 \leq j_1$$

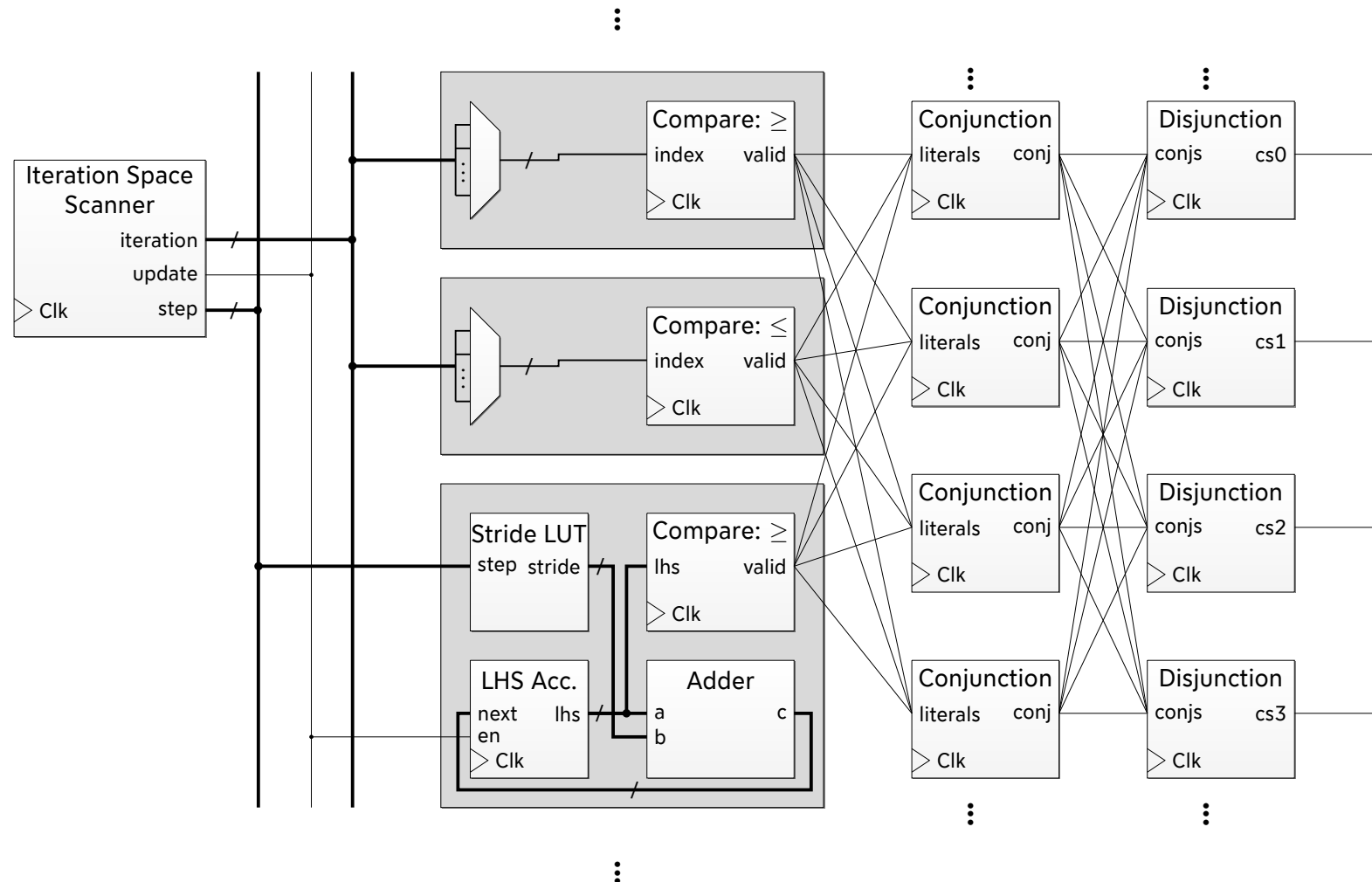
We classify loop-index comparisons into five categories:

- **Constant equality:** $j_x = a$
- **Constant lower bound:** $j_x \geq a$
- **Constant upper bound:** $j_x \leq a$
- **Affine equality:** $\mathbf{a}j = b$
- **Affine inequality:** $\mathbf{a}j \geq b$

⇒ Each unified condition is assigned one binary control signal.

⇒ The signal is high for iterations in its one domain and low otherwise.

⇒ A single Global Controller (GC) evaluates all control signals.



Control Instructions

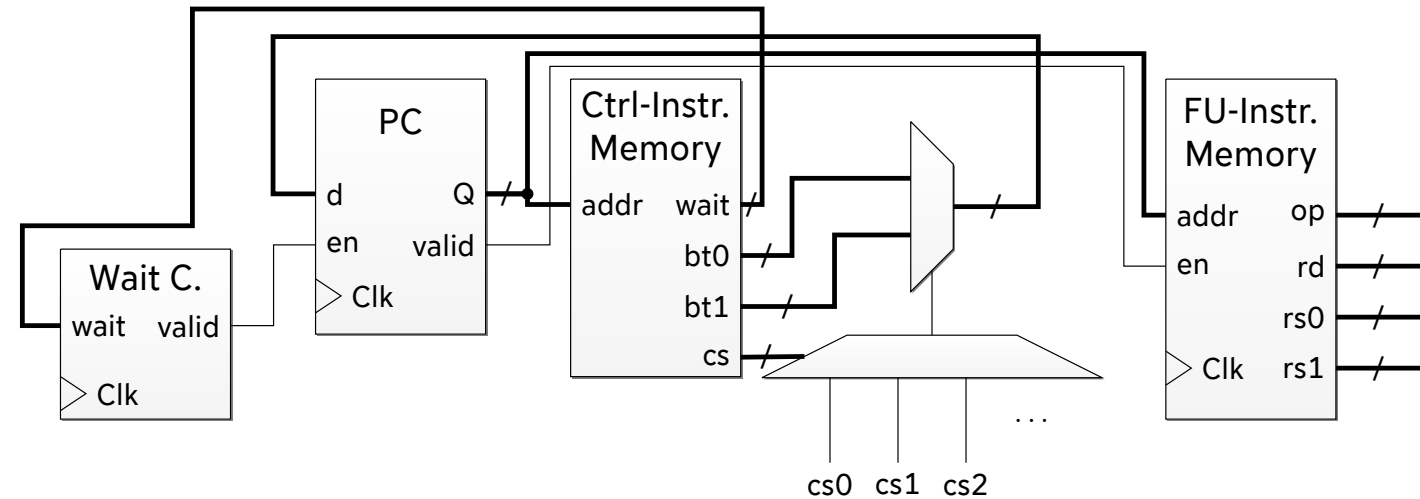
Using control signals to steer the program counter

Instruction format

```
op rd rs0 rs1  
bt0 bt1 cs wait
```

Semantics

```
rd ← rs0 op rs1  
wait for wait cycles  
if cs = 1 goto bt0  
      else goto bt1
```



- ⇒ No pipeline hazards: control and data paths are decoupled.
- ⇒ One instruction per cycle without branch prediction.
- ⇒ Instructions execute precisely in the scheduled order.

Evaluation

Control allocation and instruction-memory reduction

Loop			Scheduling				Control Allocation								Control Generation				
Benchmark	# Equations	n	Π	L_{local}	λ_0^k	λ_1^k	# Program Blocks	$ C $	$ C_{prime} $	$ C_{unified} $	# Programs	# Instructions	# Waits	# Instr. per Prog.	# Lower Bounds	# Upper Bounds	# Affine Bounds	# Conjunctions	# Disjunctions
GEMM	11	3	1	10	11	27	429	289	23	8	84	429	0	9	9	8	4	19	8
TRSM	11	3	6	20	11	151	672	584	96	13	64	976	2736	25	22	19	34	67	13
LU	13	3	29	29	179	643	799	463	119	18	48	1897	19 930	247	24	20	31	91	18
ATAx	12	2	3	12	131	38	264	150	18	10	58	350	326	14	9	6	0	19	10
MVT	13	2	3	8	23	71	513	224	17	9	72	917	478	24	5	3	1	14	9
GESMMV	12	2	3	7	8	38	168	80	7	4	52	240	160	8	5	2	0	7	4

⇒ $15 \times$ to $45 \times$ fewer control conditions across six PolyBench kernels.

⇒ Up to 91 % less instruction-memory demand through wait counters.

	Insts.	LUTs	FFs	BRAMs	DSPs
4×4 TCPA	1	220 524	205 774	656	48
Avg. Processing element (PE)	16	11 091	8563	39	3
Avg. Functional units	16	2967	3380	7	3
Avg. Data register file	16	6000	2947	2	0
Avg. Control register file	16	645	711	30	0
Avg. Interconnect	16	712	683	0	0
Avg. I/O buffer incl. AGs	4	6523	11 197	8	0
Avg. Address Generator	32	483	740	0	0
Global controller	1	9741	17 861	0	0
Loop I/O controller	1	5738	4277	4	0

- ⇒ The GC uses 4.17% of LUTs and 8.04% of FFs, roughly the cost of one PE.
- ⇒ Total loop-control overhead remains below 10% of the TCPA resources.

- **Challenge: Loop control can dominate execution time**
- **Solution: Zero-overhead loop control for Tightly Coupled Processor Arrays**
 - No runtime overhead: the theoretical schedule is executed exactly.
 - Low hardware overhead.
- **Methodology**
 - Generate control-flow graphs from the schedule.
 - Find and minimize control conditions.
 - Map them onto a shared Global Controller.
 - Generate and distribute control signals to all PEs.
 - Select the next program block using propagated control signals.

- [Han+14] F. Hannig, V. Lari, S. Boppu, A. Tanase, and O. Reiche. "Invasive Tightly-Coupled Processor Arrays: A Domain-Specific Architecture/Compiler Co-Design Approach". In: *TECS* 13.4s (2014), 133:1–133:29.
- [Kis+06] D. Kissler, F. Hannig, A. Kupriyanov, and J. Teich. "A Highly Parameterizable Parallel Processor Array Architecture". In: *FPT*. 2006, pp. 105–112.
- [Pou12] L.-N. Pouchet. *PolyBench/C: The Polyhedral Benchmark suite*. <https://web.cs.ucla.edu/~pouchet/software/polybench/>. 2012.
- [Tei+22] J. Teich, M. Brand, F. Hannig, C. Heidorn, D. Walter, and M. Witterauf. "Invasive Tightly-Coupled Processor Arrays". In: *Invasive Computing*. FAU University Press, 2022.
- [Wit+21] M. Witterauf, D. Walter, F. Hannig, and J. Teich. "Symbolic Loop Compilation for Tightly Coupled Processor Arrays". In: *TECS* 20.5 (2021), 49:1–49:31.

Thanks for listening.
Any questions?